John von Neumann Institute for Computing

Parallel Compression of 3D Meshes for Efficient Distributed Visualization

A. Clematis, D. D'Agostino, V. Gianuzzi

published in

Parallel Computing:
Current & Future Issues of High-End Computing,
Proceedings of the International Conference ParCo 2005,
G.R. Joubert, W.E. Nagel, F.J. Peters, O. Plata, P. Tirado, E. Zapata (Editors),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **33**, ISBN 3-00-017352-8, pp. 655-662, 2006.

http://www.fz-juelich.de/nic-series/volume33

Parallel Compression of 3D Meshes for Efficient Distributed Visualization

Andrea Clematis[a], Daniele D'Agostino[a], Vittoria Gianuzzi[b]

[a]IMATI-CNR, Genova, {clematis,dago}@ge.imati.cnr.it

[b]DISI, Univeristy of Genova, gianuzzi@disi.unige.it

In a distributed visualization environment transmission time is dominant because of the amount of data to be moved and the limitations of available bandwidth. In this paper we address the problem to speed up the compression operation of large Triangulated Irregular Networks (TIN) using commodity clusters. In our case TINs represent isosurfaces extracted from volumetric data sets. The proposed parallel compression algorithm is based on Edgebreaker [1], one of the most powerful connectivity compression algorithm, and it exploits mesh partitioning produced during the parallel isosurface extraction operation. In this way a high speed-up of the compression module and a considerable improvement of the visualization system are obtained.

1. Introduction

Nowadays large collections of 3D and volumetric data are available, and many people with expertise in different disciplines access these data through the Web. Isosurface extraction [2] is a basic operation that permits to implement many types of queries on volumetric data. The product of the isosurface extraction operation is a Triangulated Irregular Network (TIN) often containing a huge number of triangles, depending on the original data set and on the requested isovalue.

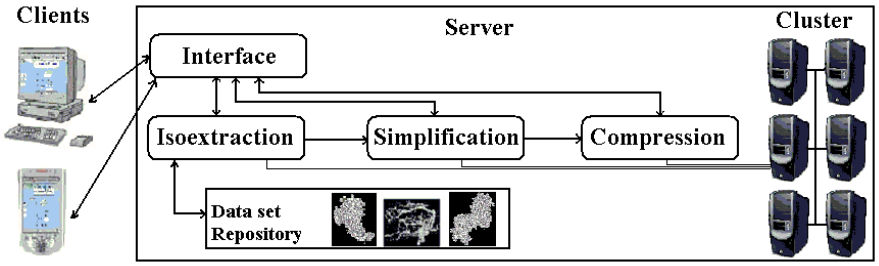

Figure 1. A data processing pipeline for remote visualization

In [3] a typical scenario is presented, where a client accesses a Web server in order to study a 3D data set, for example to visualize protein surfaces in order to define correlation between different macromolecules [4]. The computational pipeline executed on the server consists of an isosurface extraction step [5], that may be followed by a simplification step [6] for progressive visualization, and a compression step to reduce the amount of transferred data as represented in Figure 1.

Reducing the amount of transmitted data is of paramount importance for remote visualization, in particular in a Grid environment [7]. The interconnection network in fact may have variable characteristics, but even using advanced interconnection technologies, the available bandwidth may

represent the bottleneck of a visualization system [8], due to data size. Our computational pipeline doesn't make exception, because extracted isosurfaces can be very large.

The pipeline execution may benefit of parallel computing at different levels, and may have an adaptive behaviour in order to provide the best suited configuration depending on the specific visualization task. In this paper we deal with the parallelization of the compression step using a cluster of COTS PCs. In the pipeline configuration the exploitation of the data partitioning derived from the previous steps makes its execution particularly efficient, because the adopted domain partitioning strategy is finalized to obtain the best speed up for the whole pipeline [9]. However the proposed parallel algorithm can be used in different software configurations provided that it is combined with a suitable TIN partitioning algorithm.

The paper is organized in the following way: in Section 2 different approaches to compress triangular meshes are shortly described, in Section 3 Edgebreaker compression algorithm is introduced, and in Section 4 the parallel compression algorithm and the mesh partitioning strategy are outlined. In Section 5 experimental results are discussed, followed by conclusions and future works in Section 6.

2. Triangular Mesh Compression

A triangular mesh is defined by its geometry and connectivity. The *geometry* corresponds to the vertices, represented by their coordinates (with possibly other information as normal, textures ...), while the *connectivity* is represented by the triangle-vertex incidence relation.

For this reason mesh compression is considered as composed by two distinct operations, *geometry and connectivity encoding*.

Several efficient general purpose techniques were proposed to deal with geometry encoding. They are mostly based on three operations: quantization, prediction, and entropy coding. On the contrary for connectivity encoding the proposed techniques present more or less good results depending on data characteristics.

In [10] the different connectivity compression techniques are classified as *single-rate* and *progressive*. In the first case the mesh is completely encoded before the transmission and decoded after the complete reception. In the second case a coarse mesh is initially received and decoded, then it is progressively refined.

It is possible to further subdivide single-rate algorithms between *lossy* and *lossless*. Lossy algorithms as [11] after the decompression do not recreate exactly the original meshes as lossless ones do. The lossy approach may be useful when the original connectivity does not need to be preserved, so that the triangle mesh may be re-sampled to produce a more regular approximating mesh.

Progressive techniques are useful to provide the client with the possibility to visualize meshes at different levels of details. This can be useful in particular for large meshes, because the visualization of surfaces made by several millions triangles is difficult, if not impossible, even using advanced workstations. In our visualization pipeline we face the problem with the simplification component, that reduces the number of triangles, for example by merging the planar ones. Furthermore, we do not make assumption on the importance of the original connectivity. For these reasons we decided to adopt a single-rate, lossless compression algorithm, and among the interesting proposals belonging to this class as [12, 13], we chose to parallelize "Edgebreaker" [1], one of the most performant single-rate, lossless connectivity compression algorithm.

An in-depth study of this class of algorithms is beyond the goal of this paper, and the interested readers may refer to [14, 10] for a survey and to [15] for a comparison of different algorithms with Edgebreaker.

Edgebreaker is one of the best connectivity compression algorithm, because it allows to compress the connectivity of a manifold triangular mesh homeomorphic to a sphere using 1.84 bit in the worst case for each triangle. This algorithm was afterward improved to threat efficiently meshes with handles [16] and bounding loops [17] as it is the case of most isosurfaces, thus we decided to parallelize this algorithm for our computational pipeline.

Our parallel implementation of Edgebreaker permits to achieve a high speed-up, and at the same time provides a solution to process large meshes by using the aggregate memory of a COTS cluster. In fact, one key problem, common to most of the compression techniques, is that they work in core, then they are not able to process very large meshes. In [18], an out-of-core version of the Touma and Gotsman algorithm [12] is proposed, while at the best of our knowledge there are no previous works about parallel version of TIN compression algorithms.

3. The Edgebreaker Compression Algorithm

The input of the Edgebreaker algorithm is a manifold, orientable meshes that may contain handles and bounding loops.

The algorithm is a finite state machine: given a starting triangle it traverses all the other triangles one at a time along a spiral path. Each visited triangle and its vertices are marked. At the beginning, only the start triangle and its three vertices are marked. The decision of which triangle adjacent to the current one will be visited in the next step is based only on local information and five rules. These rules correspond to label each triangle with one character between ‘C’, ‘L’, ‘E’, ‘R’, ‘S’. The ‘C’ and ‘S’ cases correspond to triangles having neither the left nor the right triangles marked. The difference is that for the ‘C’ case one of the triangle vertices is unvisited. The ‘L’ and ‘R’ labels are used when respectively only the left triangle or the right triangle has already been visited. The ‘E’ case arises when all the neighbour triangles are already marked.

The complete connectivity is encoded considering the label associated to each triangle in the visiting order. The result is a string over this five-symbol alphabet named *clers* string, as exemplified in Figure 2.

Edgebreaker is a pure connectivity compression algorithm. However it is possible to combine it with the compression of the geometry, as described in [16]. The vertex coordinates are encoded in correspondence of ‘C’ triangles using the *parallelogram rule*. The result of the geometry encoding is a sequence of corrector factors plus few coordinates encoded explicitly. This sequence is called *delta* string. We consider as the output of this “complete” version of Edgebreaker both the strings representing the connectivity and the geometry. The size of these strings normally is further reduced applying a general purpose entropy encoding technique.

If a surface is composed by several connected components, that is the case for many isosurfaces, the algorithm processes each component independently, and generate a pair of (clers, delta) strings for each component.

4. The Parallel Compression Algorithm

The presented compression algorithm is a good candidate for parallel processing, in fact most of the operations performed during this process are local. In the case of different connected components we may process them in parallel in a straightforward way, since their processing is independent as noticed above. However, we may incur in load unbalancing due to the different number of triangles of each component. In order to deal with general isosurfaces, that may contain a single large component, and to ensure load balancing, we should find out a suitable mesh partitioning strategy

and verify the correctness and the quality of the algorithm. It is useful to consider initially load balancing, and then discuss correctness and quality of the parallel algorithm.

In order to balance the cost of parallel execution of Edgebreaker across different processors, it is necessary and sufficient to balance the number of triangles contained in the partitions of the mesh distributed among processors. In fact the encoding cost of the clers string is proportional to the number of triangles, while the encoding of the delta string is proportional to the number of vertices, but, following the Euler rule, the number of triangles and the number of vertices are related. To divide a TIN mesh with an approximate equal number of triangles for each partition is in general a costly operation because of the irregular structure of the mesh [19].

In our case, we exploit the information about triangles distribution across the original volumetric data set collected during the first phase of the isosurface extraction process. This information is of paramount importance to obtain a load balanced partitioning of the isosurface, that is exploited during the simplification and compression steps. The adopted partitioning strategy is based on the division of the surface along the z-axis using orthogonal slices. The information collected during the first phase of isoextraction permits to assign a processor the portion of the isosurface included between two slices in such a way that each processor will handle about the same number of triangles, and a minimal set of borders.

Let us consider now the algorithm correctness and the quality of the produced output.

The parallel algorithm is correct if the mesh generated after the decompression is equivalent to the original mesh. Our parallel implementation provides the same result of the sequential algorithm, for each partition of the mesh, provided that a unique bounding box for the quantization of the coordinates of the vertices is used by the different processes, in order to avoid topological errors after the geometry decompression. This requires a unique data exchange among processes before the quantization operation.

The quality of the algorithm is firstly measured by the achieved compression rate. Mesh partition-

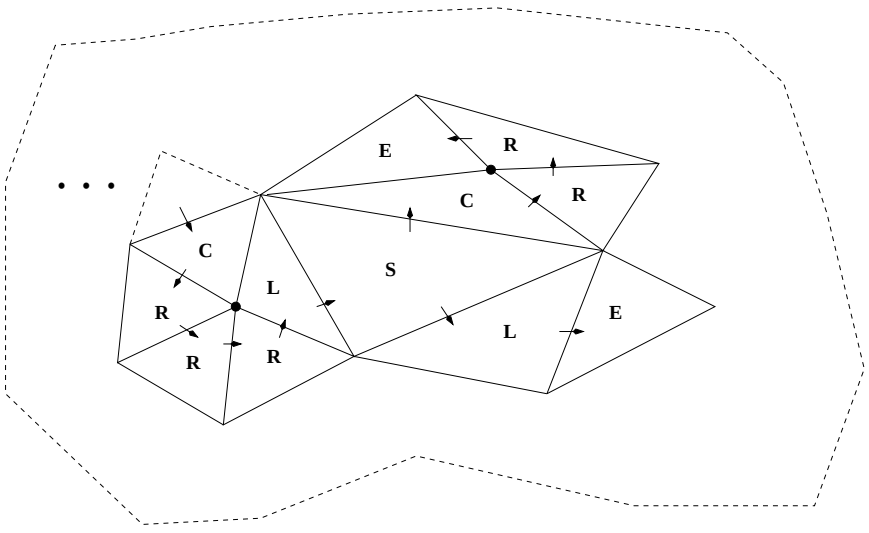

Figure 2. An example of connectivity encoding using Edgebreaker. The dashed lines represent the part of the mesh previously visited and encoded. Not visited vertices are denoted with black circles.

ing introduces new borders that do not exist in the original mesh. The new borders cause an increase in the size of the compressed mesh, mainly because of border vertices replication. Thus the parallel algorithm achieves a worst compression rate. Quantitative evaluation are provided in the next Section. In order to avoid the replication, border vertices could be kept in a shared data structure and the quantization and entropy reduction process, as well as the decompression operation, should be suitably modified. On the base of the analysis of trade-off between the saved space and increment in the computational cost we decided to accept to pay the fee of a reduced compression rate, while keeping the high scalability of the parallel algorithm.

Despite the reduced compression rate attained, the parallel algorithm has many merits:

- The parallel algorithm is highly scalable and an almost linear speed-up is achievable in many cases as shown by experimental results. This point has a large influence on the performance of the whole visualization pipeline.
- The parallel algorithm, using the aggregate memory available on the cluster, increases in a significative way the size of meshes that can be compressed. To this purpose the parallel approach seems to be more competitive with respect to the use of out-of-core algorithms because of the speed-up in the compression process. Moreover the scalability of the parallel algorithm makes it possible to handle meshes of arbitrary size, provided that enough processing units are available.
- As a side effect, the decompression operation can be afforded by almost any client at a reasonable cost, because of the reduced size of each partition.
- Despite its simple parallel implementation, the compression module is of paramount importance for the efficient and effective use of the visualization pipeline.

These merits are supported by the experimental results provided in the next Section.

5. Experimental Results

Sequential computing times have been collected using a Linux PC equipped with a 2.66 GHz Pentium processor, 512 MB of Ram and two EIDE disks interfaced in RAID 0. Parallel computing times have been collected using a cluster of 16 PCs with the previous characteristics, interconnected through an Ethernet - Gigabit switch and having input data sets stored in a PVFS 1 parallel filesystem. The sequential and parallel algorithms are based on the code freely available in [20] and [16], both are implemented in C and the MPICH library is used for parallelization.

Table 1
This table summarizes I/O data volumes of the isosurfaces extracted for different data sets and isovalues and the size of data compressed with the sequential implementation of Edgebreaker

Data set - isovalue	Input size	Triangles	Output Size	Edgebreaker
Bonsai - 2	16 MB	3,896,986	67 MB	5.7 MB
Frog - 75	30 MB	2,655,552	45 MB	3.7 MB
Xmastree - 180	499,5 MB	4,514,539	78 MB	5.7 MB
Xmastree - 52	499,5 MB	21,719,037	374 MB	31 MB
VisFemale - 1000	867 MB	69,600,216	1.2 GB	N.A.

In Table 1 the characteristics of meshes used in the tests are presented. These isosurfaces are extracted from Computed Tomography scans of a bonsai, a frog, a Christmas tree, and a female cadaver. The Christmas Tree data set was generated from a real world Christmas Tree by the Department of Radiology, University of Vienna and the Institute of Computer Graphics and Algorithms, Vienna University of Technology. The female cadaver is an anatomical data set developed under a contract from the NLM by the Departments of Cellular and Structural Biology, and Radiology, University of Colorado School of Medicine.

We can see that with Edgebreaker the size of compressed data is about 12% of the original size using a quantization on 16 bits. The compressed data size for “VisFemale - 1000” is not available because the mesh is too large for a sequential implementation.

Table 2
This table summarizes the times in seconds of the sequential and parallel version of Edgebreaker algorithm for the isosurfaces described in Table 1. In brackets the speed up values.

Data set - isovalue	Sequential	4 procs	8 procs	16 procs
Bonsai - 2	10.28	2.85 (3.6)	1.42 (7.6)	0.66 (15.5)
Frog - 75	6.81	1.84 (3.7)	0.87 (7.8)	0.52 (13.1)
Xmastree - 180	10.73	2.92 (3.7)	1.43 (7.5)	0.68 (15.8)
Xmastree - 52	796.30	22.99 (-)	6.94 (-)	3.42 (-)
VisFemale - 1000	N.A.	N.A.	N.A.	19.21 (N.A.)

In Table 2 execution times in seconds for the sequential and the parallel compression algorithm together with speedup values (in brackets) are presented. We used 4, 8 and 16 processors to execute the parallel algorithm.

The dataset “VisFemale - 1000” is manageable only using 16 processes, with an execution time of about 19 seconds. With dataset “Xmastree - 52” we got using 16 processors a speed up of about 232. In fact to treat this mesh, the sequential algorithm has to use the virtual memory of the system, with a considerable performance degradation due to frequent page faults. Thus we do not consider the obtained super linear speed-up for this and similar cases. However these results emphasize the importance of parallel compression.

As said before, to evaluate the effectiveness of our parallelization strategy it is necessary to take into account both the speed up achieved and the quality of the results, represented by the size of the entropy encoding of the “delta” and the “clers” strings.

Table 3
This table compares the size of the compresses data resulting from the sequential algorithm, the parallel algorithm executed using 16 processes, and GNU Gzip

Data set - isovalue	Sequential EB	Parallel EB	Gzip
Bonsai - 2	5.7 MB	6.3 MB	23 MB
Frog - 75	3.7 MB	4 MB	16 MB
Xmas tree - 180	5.7 MB	6.1 MB	28 MB
Xmas tree - 52	31 MB	32 MB	134 MB
VisFemale - 1000	N.A.	112 MB	443 MB

In Table 3 the incidence of the duplicated boundary vertices is presented in the worst case, when each isosurface is partitioned among 16 processors. As we can see, the data size is increased at most of 10% . In practice, the degradation due to vertex duplication has not an heavy impact from the result size point of view, while the speed up achievable is nearly linear.

In the same table we present a comparison between the size of the data compressed using Edgebreaker and a general purpose technique, GNU Gzip, to demonstrate how a specific compression technique is more effective.

To better appreciate the effectiveness of the use of the parallel compression we consider the total time needed to transmit isosurface from the server to a client in three different cases:

1. the transmission is done without compression;
2. sequential compression is adopted, and decompression is executed on the client side;
3. parallel compression is adopted, and decompression followed by a mesh sewing is executed by the client.

The client is a Linux PC with the same characteristics of the the node of the cluster, while the transmission is on a geographic network connecting our institute in Genoa with the Institute for Biotecnologies of CNR in Milan, and the network bandwidth is of about 143KB/sec.

The measured total time for the Xmas tree - 52 is 44min:37sec. without compression, 17min:36sec. with sequential compression and 4min:31sec with parallel compression.

6. Conclusion and Future Works

The main contribution of this paper is a data parallel compression algorithm for TINs based on the Edgebreaker connectivity compression algorithm.

Despite the simple approach, our algorithm presents two advantages. The first is the high speed up obtainable, because each process encodes its data independently. The second is the possibility to efficiently compress meshes of arbitrary size using a cluster of PCs with a sufficient amount of aggregate memory. The decompression remains a sequential operation, but is able to manage the produced results, because they are provided in a partitioned form.

A drawback is represented by the degradation of the quality of the results that, for a compression algorithm, is represented by the size of the compressed data. This increment is of about 10% so, considering also the achieved near linear speed up, we can conclude that the proposed algorithm represents a good compromise between the speed up achievable and the quality of the results.

We plan to develop an ad-hoc mesh partitioning algorithm and to improve the algorithm in order to reduce the impact of duplicated vertices.

7. Acknowledgments

This work has been supported by MIUR programme L.449/97-00 High Performance Distributed Computing Platform, and by FIRB strategic project on Enabling Technologies for Information Society, Grid.it.

References

[1] J. Rossignac: Edgebreaker: Connectivity compression for triangle meshes. IEEE Transactions on Visualization and Computer Graphics, Vol. 5, No. 1, pp. 47-61. 1999.

[2] W.E. Lorensen, H.E. Cline: Marching Cubes: a high resolution 3D surface reconstruction algorithm. Computer Graphics, Vol. 21, No. 4, pp. 163-169. 1987.

[3] A. Clematis, D. D'Agostino, W. De Marco and V. Gianuzzi: A Web-Based Isosurface Extraction System for Heterogeneous Clients. Proceedings of the 29th Euromicro Conference, IEEE Computer Society Press, pp. 148-156. 2003.

[4] I. Merelli, Luciano Milanesi, Daniele D'Agostino, Andrea Clematis, Marco Vanneschi, Marco Danelutto: Using Parallel Isosurface Extraction in Superficial Molecular Modeling. Proceedings of the 1st International Conference on Distributed Frameworks for Multimedia Applications (DFMA 2005), IEEE Computer Society, pp. 288-294. 2005.

[5] A. Clematis, D. D'Agostino, V. Gianuzzi: An Online Parallel Algorithm for Remote Visualization of Isosurfaces. Proceedings of the 10th EURO PVM MPI Conference, LNCS no. 2840, pp. 160-169. 2003.

[6] A. Clematis, D. D'Agostino, V. Gianuzzi, M. Mancini: Parallel Decimation of 3D Meshes for Efficent Web based Isosurface Extraction. Proceedings of the International Conference of Parallel Computation (PARCO 2003), Advances in Parallel Computing No. 13, pp. 159-166. 2004.

[7] P. Heinzlreiter, D. Kranzlmüller: Visualization Services on the Grid: The Grid Visualization Kernel. Parallel Processing Letters, Vol. 13, No. 2, pp. 135-148. 2003.

[8] S. Fisher: Berkeley Lab Proves 10-Gigabit Ethernet Data Transfer is a Reality. Supercomputing Online. 3 Jul. 2003.

[9] A. Clematis, D. D'Agostino, V. Gianuzzi: Load Balancing and Computing Strategies in Pipeline Optimization for Parallel Visualization of 3D Irregular Meshes. Accepted to the 12th EURO PVM MPI Conference. Sorrento (Naples), ITALY, September 18/21, 2005

[10] P. Alliez and C. Gotsman: Recent Advances in Compression of 3D Meshes. Proceedings of the Symposium on Multiresolution in Geometric Modeling, 2003.

[11] M. Attene, B. Falcidieno, M. Spagnuolo and J. Rossignac: SwingWrapper: Retiling Triangle Meshes for Better Compression. ACM Transactions in Graphics, Vo. 22, No. 4, pp. 982-996, 2003.

[12] C. Touma and C. Gotsman. Triangle Mesh Compression. Graphics Interface 98 Conference Proceedings, pp. 26-34, 1998.

[13] S. Gumhold and W. Straßer. Real time compression of triangle mesh connectivity. Proceedings of SIGGRAPH '98, pp. 133-140, 1998.

[14] C. Gotsman, S. Gumhold, and L. Kobbelt: Simplification and Compression of 3D Meshes. Tutorials on Multiresolution in Geometric Modelling, A. Iske, E. Quak, and M.S. Floater (eds.), Springer-Verlag, pp. 319-361, 2002.

[15] J. Rossignac: 3D Mesh Compression Chapter in The visualization handbook, C. D. Hansen and C. R. Johnson (eds.), Academic Press, 2004.

[16] A. Szymczak, D. King, J. Rossignac: An Edgebreaker-based efficient compression scheme for regular meshes. Computational Geometry, Vol. 20, No. 1-2, pp. 53-68. 2001.
The code is available at http://www.cc.gatech.edu/fac/Andrzej.Szymczak/eb.html

[17] T. Lewiner, H. Lopes, J. Rossignac, A. Wilson-Vieira: Efficient Edgebreaker for surfaces of arbitrary topology. Proceedings of 17th Brazilian Symposium on Computer Graphics and Image Processing, (SIBGRAPI'04), pp. 218-225. 2004.

[18] M. Isenburg, S. Gumhold: Out-of-core compression for gigantic polygon meshes ACM Transactions on Graphics, vol 22, no. 3, pp. 935-942, 2003.

[19] C. Walshaw, M. Cross, and M.G. Everett, Parallel Dynamic Graph Partitioning for Adaptive Unstructured Meshes Journal of Parallel and Distributed Computing, 47, pp. 102-108, 1997.

[20] Edgebreaker source code: http://www.gvu.gatech.edu/ jarek/edgebreaker/eb/